;TATION PAGE		Form Approved OPM No. 0704-0188 (2)
AD-A226 899	1 hour per response, including the time for reviewing instructions, searching existing data sources gathering and comments regarding this burden estimate or any other aspect of this collection of information, including suggestions Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to dget, Washington, DC 20503.	
1.	T DATE	3. REPORT TYPE AND DATES COVERED Final 10 May 1990 to 10 May 1991
4. TITLE AND SUBTITLE Ada Compiler Validation Summary Report: Concurrent Computer Corporation, C^3Ada Version 0.5, Concurrent Computer Corporation 8400 with MIPS/R3000 CPU and MIPS/3010 Floating Point under RTU Version 5.1 (Host & Target), 900427I1.11008		5. FUNDING NUMBERS
6. AUTHOR(S) IABG-AVF Ottobrunn, FEDERAL REPUBLIC OF GERMANY		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) IABG-AVF, Industrieanlagen-Betriebsgeselschaft Dept. SZT Einsteinstrasse 20 D-8012 Ottobrunn FEDERAL REPUBLIC OF GERMANY		8. PERFORMING ORGANIZATION REPORT NUMBER IABG-VSR-071
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Ada Joint Program Office United States Department of Defense Washington, D.C. 20301-3081		10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES		
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.		12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Concurrent Computer Corporation, C^3Ada Version 0.5, Ottobrunn, West Germany, Concurrent Computer Corporation 8400 with MIPS/R3000 CPU and MIPS/3010 Floating Point under RTU Version 5.1 (Host & Target), ACVC 1.11.

14. SUBJECT TERMS Ada programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, Validation Testing, Ada Validation Office, Ada Validation Facility, ANSI/MIL-STD-1815A, Ada Joint Program Office			15. NUMBER OF PAGES
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT

90 09 [illegible]

CHAPTER 1

INTRODUCTION

This Validation Summary Report ~~(VSR)~~ describes the extent to which a specific Ada compiler conforms to the Ada Standard, ANSI/MIL-STD-1815A. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability ~~(ACVC)~~. An Ada compiler must be implemented according to the Ada Standard, and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from the characteristics of particular operating systems, hardware, or implementation strategies. All the dependencies observed during the process of testing this compiler are given in this report.

The information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation-dependent but is permitted by the Ada Standard. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, and during execution.

(KR)

AVF Control Number: IABG-VSR-071
10 May 1990

Ada COMPILER VALIDATION SUMMARY REPORT:

Certificate Number: #900427I1.11008
Concurrent Computer Corporation
C^{3}Ada Version 0.5
Concurrent Computer Corporation 8400
with MIPS/R3000 CPU and MIPS/3010 Floating Point
under RTU Version 5.1

Prepared By:
IABG mbH, Abt. ITE
Einsteinstrasse 20
D-8012 Ottobrunn
West Germany

Certificate Information

The following Ada implementation was tested and determined to pass ACVC 1.11. Testing was completed on 27 April 1990.

Compiler Name and Version:	C^3Ada Version 0.5
Host Computer System:	Concurrent Computer Corporation 8400 with MIPS/R3000 CPU and MIPS/3010 Floating Point under RTU Version 5.1
Target Computer System:	Same as Host

A more detailed description of this Ada implementation is found in section 3.1 of this report.

As a result of this validation effort, Validation Certificate #900427I1.11008, is awarded to Concurrent Computer Corporation. This certificate expires on 01 June 1992.

This report has been reviewed and is approved.

Michael Tonndorf

IABG mbH, Abt. ITE
Michael Tonndorf
Einsteinstrasse 20
D-8012 Ottobrunn
West Germany

[illegible]

Ada Validation Organization
Director, Computer & Software Engineering Division
Institute for Defense Analyses
Alexandria VA 22311

[illegible]

Ada Joint Program Office
Dr. John Solomond
Director
Department of Defense
Washington DC 20301

DECLARATION OF CONFORMANCE

Customer:	Concurrent Computer Corporation
Ada Validation Facility:	IABG, Federal Republic of Germany
ACVC Version:	1.11

Ada Implementation:

Compiler Name and Version:	C ^{3}Ada **Version:** 0.5
Host Computer System:	Concurrent Computer Corporation 8400 with MIPS/R3000 CPU and MIPS/3010 Floating point under RTU Version 5.1
Target Computer System:	Same as Host

Customer's Declaration

I, the undersigned, representing Concurrent Computer Corporation, declare that Concurrent Computer Corporation has no knowledge of deliberate deviations from the Ada Language Standard ANSI/MIL-STD-1815A in the implementation listed in this declaration. I declare that Systeam KG is the Implementor of the above implementation and the certificates shall be awarded in the name of Concurrent Computer Corporation's corporate name.

[illegible] 5/21/90
Seetharama Shastry (date)
Senior Manager, System Software Development

[illegible] 5/21/90
Dr. Georg Winterstein (date)
President, Systeam KG

CONTENTS

CHAPTER 1

INTRODUCTION

The Ada implementation described above was tested according to the Ada Validation Procedures [Pro89] against the Ada Standard [Ada83] using the current Ada Compiler Validation Capability (ACVC). This Validation Summary Report (VSR) gives an account of the testing of this Ada implementation. For any technical terms used in this report, the reader is referred to [Pro89]. A detailed description of the ACVC may be found in the current ACVC User's Guide [UG89].

1.1 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the Ada Certification Body may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject implementation has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from the AVF which preformed this validation or from:

National Technical Information Service
5285 Port Royal Road
Springfield VA 22161

Questions regarding this report or the validation test results should be directed to the AVF which performed this validation or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311

1.2 REFERENCES

[Ada83] Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.

[Pro89] Ada Compiler Validation Procedures, Version 2.0, Ada Joint Program Office, May 1989.

[UG89] Ada Compiler Validation Capability User's Guide, 24 October 1989.

1.3 ACVC TEST CLASSES

Compliance of Ada implementations is tested by means of the ACVC. The ACVC contains a collection of test programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable. Class B and class L tests are expected to produce errors at compile time and link time, respectively.

The executable tests are written in a self-checking manner and produce a PASSED, FAILED, or NOT APPLICABLE message indicating the result when they are executed. Three Ada library units, the packages REPORT and SPPRT13, and the procedure CHECK_FILE are used for this purpose. The package REPORT also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The package SPPRT13 is used by many tests for Chapter 13 of the Ada Standard. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for Chapter 14 of the Ada Standard. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. If these units are not operating correctly, validation testing is discontinued.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that all violations of the Ada Standard are detected. Some of the class B tests contain legal Ada code which must not be flagged illegal by the compiler. This behavior is also verified.

Class L tests check that an Ada implementation correctly detects violation of the Ada Standard involving multiple, separately compiled units. Errors are expected at link time, and execution is attempted.

In some tests of the ACVC, certain macro strings have to be replaced by implementation-specific values -- for example, the largest integer. A list of the values used for this implementation is provided in Appendix A. In addition to these anticipated test modifications, additional changes may be required to remove unforeseen conflicts between the tests and implementation-dependent characteristics. The modifications required for this implementation are described in section 2.3.

For each Ada implementation, a customized test suite is produced by the AVF. This customization consists of making the modifications described in the preceding paragraph, removing withdrawn tests (see section 2.1) and, possibly some inapplicable tests (see Section 3.2 and [UG89]).

In order to pass an ACVC an Ada implementation must process each test of the customized test suite according to the Ada Standard.

1.4 DEFINITION OF TERMS

Ada Compiler	The software and any needed hardware that have to be added to a given host and target computer system to allow transformation of Ada programs into executable form and execution thereof.
Ada Compiler Validation Capability (ACVC)	The means for testing compliance of Ada implementations, consisting of the test suite, the support programs, the ACVC user's guide and the template for the validation summary report.
Ada Implementation	An Ada compiler with its host computer system and its target computer system.
Ada Validation Facility (AVF)	The part of the certification body which carries out the procedures required to establish the compliance of an Ada implementation.
Ada Validation Organization (AVO)	The part of the certification body that provides technical guidance for operations of the Ada certification system.
Compliance of an Ada Implementation	The ability of the implementation to pass an ACVC version.
Computer System	A functional unit, consisting of one or more computers and associated software, that uses common storage for all or part of a program and also for all or part of the data necessary for the execution of the program; executes user-written or user-designated programs; performs user-designated data manipulation, including arithmetic operations and logic operations; and that can execute programs that modify themselves during execution. A computer system may be a stand-alone unit or may consist of several inter-connected units.
Conformity	Fulfillment by a product, process or service of all requirements specified.
Customer	An individual or corporate entity who enters into an agreement with an AVF which specifies the terms and

	conditions for AVF services (of any kind) to be performed.
Declaration of Conformance	A formal statement from a customer assuring that conformity is realized or attainable on the Ada implementation for which validation status is realized.
Host Computer System	A computer system where Ada source programs are transformed into executable form.
Inapplicable test	A test that contains one or more test objectives found to be irrelevant for the given Ada implementation.
Operating System	Software that controls the execution of programs and that provides services such as resource allocation, scheduling, input/output control, and data management. Usually, operating systems are predominantly software, but partial or complete hardware implementations are possible.
Target Computer System	A computer system where the executable form of Ada programs are executed.
Validated Ada Compiler	The compiler of a validated Ada implementation.
Validated Ada Implementation	An Ada implementation that has been validated successfully either by AVF testing or by registration [Pro89].
Validation	The process of checking the conformity of an Ada compiler to the Ada programming language and of issuing a certificate for this implementation.
Withdrawn test	A test found to be incorrect and not used in conformity testing. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains erroneous or illegal use of the Ada programming language.

CHAPTER 2

IMPLEMENTATION DEPENDENCIES

2.1 WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 65 tests had been withdrawn by the Ada Validation Organization (AVO) at the time of validation testing. The rationale for withdrawing each test is available from either the AVO or the AVF. The publication date for this list of withdrawn tests is 90-03-23.

E28005C	C34006D	B41308B	C45114A	C45612B	C45651A
C46022A	B49008A	A74006A	B83022B	B83022H	B83025B
B83025D	B83026B	C83026A	C83041A	C97116A	C98003B
BA2011A	CB7001A	CB7001B	CB7004A	CC1223A	BC1226A
CC1226B	BC3009B	CD2A21E	CD2A23E	CD2A32A	CD2A41A
CD2A41E	CD2A87A	CD2B15C	BD3006A	CD4022A	CD4022D
CD4024B	CD4024C	CD4024D	CD4031A	CD4051D	CD5111A
CD7004C	ED7005D	CD7005E	AD1B08A	AD7006A	CD7006E
AD7201A	AD7201E	CD7204B	BD8002A	BD8004C	CD9005A
CD9005B	CDA201E	CE2107I	CE2119B	CE3111C	CE3118A
CE3411B	CE3412B	CE3812A	CE3814A	CE3902B	

2.2 INAPPLICABLE TESTS

A test is inapplicable if it contains test objectives which are irrelevant for a given Ada implementation. The inapplicability criteria for some tests are explained in documents issued by ISO and the AJPO known as Ada Issues and commonly referenced in the format AI-dddd. For this implementation, the following tests were inapplicable for the reasons indicated; references to Ada Issues are included as appropriate.

The following 201 tests have floating-point type declarations requiring more digits than SYSTEM.MAX_DIGITS:

C24113L..Y (14 tests)	C35705L..Y (14 tests)

C35706L..Y (14 tests)	C35707L..Y (14 tests)
C35708L..Y (14 tests)	C35802L..Z (15 tests)
C45241L..Y (14 tests)	C45321L..Y (14 tests)
C45421L..Y (14 tests)	C45521L..Z (15 tests)
C45524L..Z (15 tests)	C45621L..Z (15 tests)
C45641L..Y (14 tests)	C46012L..Z (15 tests)

C34007P and C34007S are expected to raise CONSTRAINT_ERROR. This implementation optimizes the code at compile time on line 207 and 223 respectively, thus avoiding the operation which would raise CONSTRAINT_ERROR and so no exception is raised.

The following 21 tests check for the predefined type LONG_INTEGER:

C35404C	C45231C	C45304C	C45411C	C45412C
C45502C	C45503C	C45504C	C45504F	C45611C
C45612C	C45613C	C45614C	C45631C	C45632C
B52004D	C55B07A	B55B09C	B86001W	C86006C
CD7101F				

C35702A, C35713B, C45423B, B86001T, and C86006H check for the predefined type SHORT_FLOAT.

C35713D and B86001Z check for a predefined floating-point type other than FLOAT, SHORT_FLOAT or LONG_FLOAT.

C41401A is expected to raise CONSTRAINT_ERROR for the evaluation of certain attributes, however this implementation derives the values from the subtypes of the prefix at compile time as allowed by LRM 11.6(7). Therefore elaboration of the prefix is not involved and CONSTRAINT_ERROR is not raised.

C45346A declares an array of length INTEGER'LAST/2 + 1. This implementation raises the proper exception when the array is declared.

C45423A checks that the proper exception is raised if MACHINE_OVERFLOWS is TRUE for the floating point type FLOAT. For this implementation, MACHINE_OVERFLOWS is FALSE.

C45423B checks that the proper exception is raised if MACHINE_OVERFLOWS is TRUE for the floating point type SHORT_FLOAT. For this implementation, MACHINE_OVERFLOWS is FALSE.

C45523A and C45622A check that the proper exception is raised if MACHINE_OVERFLOWS is TRUE for floating point types with digits 5. For this implementation, MACHINE_OVERFLOWS is FALSE.

C45531M..P (4 tests) and C45532M..P (4 tests) check fixed-point-operations for types that require a SYSTEM.MAX_MANTISSA of 47 or greater.

C86001F recompiles package SYSTEM, making package TEXT_IO, and hence package REPORT, obsolete. For this implementation, the package TEXT_IO is dependent upon package SYSTEM.

B86001Y checks for a predefined fixed-point type other than DURATION.

C96005B checks for values of type DURATION'BASE that are outside the range of DURATION. There are no such values for this implementation.

CD1009C uses a representation clause specifying a non-default size for a floating-point type.

CD2A84A, CD2A84E, CD2A84I..J (2 tests), and CD2A84O use representation clauses specifying non-default sizes for access types.

CD2B15B checks that STORAGE_ERROR is raised when the storage size specified for a collection is too small to hold a single value of the designated type. For this implementation, the allocated collection size exceeds what was specified in the length clause (cf. AI-00558).

BD8001A, BD8003A, BD8004A..B (2 tests), and AD8011A use machine code insertions.

The 21 tests listed in the following table are not applicable because the given file operations are supported for the given combination of mode and file access method.

Test	File Operation	Mode	File Access Method
CE2102D	CREATE	IN_FILE	SEQUENTIAL_IO
CE2102E	CREATE	OUT_FILE	SEQUENTIAL_IO
CE2102F	CREATE	INOUT_FILE	DIRECT_IO
CE2102I	CREATE	IN_FILE	DIRECT_IO
CE2102J	CREATE	OUT_FILE	DIRECT_IO
CE2102N	OPEN	IN_FILE	SEQUENTIAL_IO
CE2102O	RESET	IN_FILE	SEQUENTIAL_IO
CE2102P	OPEN	OUT_FILE	SEQUENTIAL_IO
CE2102Q	RESET	OUT_FILE	SEQUENTIAL_IO
CE2102R	OPEN	INOUT_FILE	DIRECT_IO
CE2102S	RESET	INOUT_FILE	DIRECT_IO
CE2102T	OPEN	IN_FILE	DIRECT_IO
CE2102U	RESET	IN_FILE	DIRECT_IO
CE2102V	OPEN	OUT_FILE	DIRECT_IO
CE2102W	RESET	OUT_FILE	DIRECT_IO
CE3102E	CREATE	IN_FILE	TEXT_IO
CE3102F	RESET	Any Mode	TEXT_IO
CE3102G	DELETE	--------	TEXT_IO
CE3102I	CREATE	OUT_FILE	TEXT_IO
CE3102J	OPEN	IN_FILE	TEXT_IO
CE3102K	OPEN	OUT_FILE	TEXT_IO

CE2107C, CE2107D, CE2107L, and CE2108B attempt to associate names with temporary sequential files. The proper exception is raised when such an association is attempted.

CE2107H and CE2108D attempt to associate names with temporary direct files. The proper exception is raised when such an association is attempted.

CE2203A checks that WRITE raises USE_ERROR if the capacity of the external file is exceeded for SEQUENTIAL_IO. This implementation does not restrict file capacity.

EE2401D contains instantiations of package DIRECT_IO with unconstrained array types. This implementation raises USE_ERROR upon creation of such a file.

CE2403A checks that WRITE raises USE_ERROR if the capacity of the external file is exceeded for DIRECT_IO. This implementation does not restrict file capacity.

CE3111B and CE3115A assume that a PUT operation writes data to an external file immediately. This implementation uses line buffers; only complete lines are written to an external file by a PUT_LINE operation. Thus attempts to GET data before a PUT_LINE operation in these tests raise END_ERROR.

CE3112B assumes that temporary text files are given names. For this implementation, temporary text files are not given names.

CE3202A assumes that the NAME operation is supported for STANDARD_INPUT and STANDARD_OUTPUT. For this implementation the underlying operating system does not support the NAME operation for STANDARD_INPUT and STANDARD_OUTPUT. Thus the calls of the NAME operation for the standard files in this test raise USE_ERROR.

CE3304A checks that USE_ERROR is raised if a call to SET_LINE_LENGTH or SET_PAGE_LENGTH specifies a value that is inappropriate for external files. This implementation does not have inappropriate values for either line length or page length.

CE3413B checks that PAGE raises LAYOUT_ERROR when the value of the page number exceeds COUNT'LAST. For this implementation, the value of COUNT'LAST is greater than 150000 making the checking of this objective impractical.

2.3 TEST MODIFICATIONS

Modifications (see section 1.3) were required for 17 tests.

The following tests were split into two or more tests because this implementation did not report the violations of the Ada Standard in the way expected by the original tests.

B22003A	B24009A	B29001A	B38003A	B38009A	B38009B
B91001H	BC2001D	BC2001E	BC3204B	BC3205B	BC3205D

For the following tests a pragma ELABORATE for the package REPORT was added.

C83030C	C86007A

The following tests compile without error, as allowed by AI-00256 --the units are illegal only with respect to units that they do not depend on. However, all errors are detected at link time. The AVO ruled that this is acceptable behavior.

BC3204C	BC3204D	BC3205C	BC3205D

CHAPTER 3

PROCESSING INFORMATION

3.1 TESTING ENVIRONMENT

The Ada implementation tested in this validation effort is described adequately by the information given in the initial pages of this report.

For a point of contact for technical information about this Ada implementation system, see:

Mr. Seetharama Shastry
106 Apple Street
Tinton Falls, NJ 07724
Tel. 201-758-7277

For a point of contact for sales information about this Ada implementation system, see:

Mr. Mike Devlin
106 Apple Street
Tinton Falls, NJ 07724
Tel. 201-758-7531

Testing of this Ada implementation was conducted at Systeam KG Dr. Winterstein, Karlsruhe, Federal Republic of Germany.

3.2 TEST EXECUTION

Version 1.11 of the ACVC comprises 4140 tests. When this compiler was tested, the tests listed in section 2.1 had been withdrawn because of test errors. The AVF determined that 296 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing except for 201 executable tests that use floating-point precision exceeding that supported by the implementation. In addition, the modified tests mentioned in section 2.3 were also processed.

A magnetic tape containing the customized test suite (see section 1.3) was taken on-site by the validation team for processing. The contents of the magnetic tape were loaded onto a SUN computer and copied onto the host computer via ethernet.

After the test files were loaded onto the host computer, the full set of tests was processed by the Ada implementation.

Testing was performed using command scripts provided by the customer and reviewed by the validation team. See Appendix B for a complete listing of the processing options for this implementation. It also indicates the default options.

Tests were compiled using the comand

```
sas compile -v -l <file name>
```

and linked using the command

```
sas link -v <test name>
```

For some tests which report errors at link time a listing was created by the linker using the command

```
sas link -v -L <test name>.m <name of main>
```

The options explicitly invoked are described as follows

-l	This option controls the generation of source listings. The default action is not to generate the complete source listings.
-L	This option specifies the name of the file or the directory for the listing file. By default the warning and error messages are directed to stdout.
-v	This option causes the compiler or the linker to produce the version and information messages to be displayed. The default action is to suppress the display of such information.

Test output, compiler and linker listings, and job logs were captured on magnetic tape and archived at the AVF. The listings examined on-site by the validation team were also archived.

APPENDIX A

MACRO PARAMETERS

This appendix contains the macro parameters used for customizing the ACVC. The meaning and purpose of these parameters are explained in [UG89]. The following macro parameters are defined in terms of the value V of $MAX_IN_LEN which is the maximum input line length permitted for the tested implementation. For these parameters, Ada string expressions are given rather than the macro values themselves.

Macro Parameter	Macro Value
$BIG_ID1	(1..V-1 => 'A', V => '1')
$BIG_ID2	(1..V-1 => 'A', V => '2')
$BIG_ID3	(1..V/2 => 'A') & '3' & (1..V-1-V/2 => 'A')
$BIG_ID4	(1..V/2 => 'A') & '4' & (1..V-1-V/2 => 'A')
$BIG_INT_LIT	(1..V-3 => '0') & "298"
$BIG_REAL_LIT	(1..V-5 => '0') & "690.0"
$BIG_STRING1	'"' & (1..V/2 => 'A') & '"'
$BIG_STRING2	'"' & (1..V-1-V/2 => 'A') & '1' & '"'
$BLANKS	(1..V-20 => ' ')
$MAX_LEN_INT_BASED_LITERAL	"2:" & (1..V-5 => '0') & "11:"
$MAX_LEN_REAL_BASED_LITERAL	"16:" & (1..V-7 => '0') & "F.E:"
$MAX_STRING_LITERAL	'"' & (1..V-2 => 'A') & '"'

The following table contains the values for the remaining macro parameters.

Macro Parameter	Macro Value
$MAX_IN_LEN	255
$ACC_SIZE	32
$ALIGNMENT	4
$COUNT_LAST	2147483647
$DEFAULT_MEM_SIZE	2147483648
$DEFAULT_STOR_UNIT	8
$DEFAULT_SYS_NAME	MIPS_RTU
$DELTA_DOC	2#1.0#E-31
$ENTRY_ADDRESS	SYSTEM.INTERRUPT_VECTOR (SYSTEM.SIGUSR1)
$ENTRY_ADDRESS1	SYSTEM.INTERRUPT_VECTOR (SYSTEM.SIGUSR2)
$ENTRY_ADDRESS2	SYSTEM.INTERRUPT_VECTOR (SYSTEM.SIGUSR3)
$FIELD_LAST	512
$FILE_TERMINATOR	' '
$FIXED_NAME	NO_SUCH_FIXED_NAME
$FLOAT_NAME	NO_SUCH_FLOAT_NAME
$FORM_STRING	""
$FORM_STRING2	"CANNOT_RESTRICT_FILE_CAPACITY"
$GREATER_THAN_DURATION	0.0
$GREATER_THAN_DURATION_BASE_LAST	200_000.0
$GREATER_THAN_FLOAT_BASE_LAST	16#1.0#E+32
$GREATER_THAN_FLOAT_SAFE_LARGE	16#0.8#E+32
$GREATER_THAN_SHORT_FLOAT_SAFE_LARGE	0.0

$HIGH_PRIORITY	15
$ILLEGAL_EXTERNAL_FILE_NAME1	/nodir/file1
$ILLEGAL_EXTERNAL_FILE_NAME2	/wrongdir/file2
$INAPPROPRIATE_LINE_LENGTH	-1
$INAPPROPRIATE_PAGE_LENGTH	-1
$INCLUDE_PRAGMA1	PRAGMA INCLUDE ("A28006D1.TST")
$INCLUDE_PRAGMA2	PRAGMA INCLUDE ("B28006F1.TST")
$INTEGER_FIRST	-2147483648
$INTEGER_LAST	2147483647
$INTEGER_LAST_PLUS_1	2147483648
$INTERFACE_LANGUAGE	ASSEMBLER
$LESS_THAN_DURATION	-0.0
$LESS_THAN_DURATION_BASE_FIRST	-200_000.0
$LINE_TERMINATOR	ASCII.LF
$LOW_PRIORITY	0
$MACHINE_CODE_STATEMENT	NULL;
$MACHINE_CODE_TYPE	NO_SUCH_TYPE
$MANTISSA_DOC	31
$MAX_DIGITS	15
$MAX_INT	2147483647
$MAX_INT_PLUS_1	2147483648
$MIN_INT	-2147483648
$NAME	SHORT_SHORT_INTEGER
$NAME_LIST	MIPS_RTU

$NAME_SPECIFICATION1	/ben1/mp183/acvc11/chape/X2120A.;1
$NAME_SPECIFICATION2	/ben1/mp183/acvc11/chape/X2120B.;1
$NAME_SPECIFICATION3	/ben1/mp183/acvc11/chape/X3119A.;1
$NEG_BASED_INT	16#FFFFFFFE#
$NEW_MEM_SIZE	2147483648
$NEW_STOR_UNIT	8
$NEW_SYS_NAME	MIPS_RTU
$PAGE_TERMINATOR	' '
$RECORD_DEFINITION	NEW INTEGER
$RECORD_NAME	NO_SUCH_MACHINE_CODE_TYPE
$TASK_SIZE	32
$TASK_STORAGE_SIZE	10240
$TICK	1.0/60.0
$VARIABLE_ADDRESS	GET_VARIABLE_ADDRESS
$VARIABLE_ADDRESS1	GET_VARIABLE_ADDRESS1
$VARIABLE_ADDRESS2	GET_VARIABLE_ADDRESS2
$YOUR_PRAGMA	RESIDENT

APPENDIX B

COMPILATION SYSTEM OPTIONS

The compiler and linker options of this Ada implementation, as described in this Appendix, are provided by the customer. This list is identical to the set of options given in 3.2.

Compiler and Linker Options

-l	This option controls the generation of source listings. The default action is not to generate the complete source listings.
-L	This option specifies the name of the file or the directory for the listing file. By default, the warning and error messages are directed to `stdout`.
-v	This option causes the Compiler or the Linker to produce the version and information messages to be displayed. The default action is to suppress the display of such information.

APPENDIX C

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in Chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of this Ada implementation, as described in this Appendix, are provided by the customer. Unless specifically noted otherwise, references in this Appendix are to compiler documentation and not to this report. Implementation-specific portions of the package STANDARD, which are not a part of Appendix F, are:

```
package STANDARD is
    ..........
    type INTEGER is range -2_147_483_648  ..  2_147_483_647;

    type SHORT_INTEGER is range -32_768 .. 32_767;

    type SHORT_SHORT_INTEGER is range -128 .. 127;

    type FLOAT is digits 6 range
              - 16#0.FFFF_FF#E+32 .. 16#0.FFFF_FF#E+32;

    type LONG_FLOAT is digits 15 range
              - 16#0.FFFF_FFFF_FFFF_F8#E+256 ..
                16#0.FFFF_FFFF_FFFF_F8#E+256;

    type DURATION is delta 2#1.0#E-14 range
              - 131_072.0 .. 131_071.999_938_964_843_75;
    ..........
end STANDARD;
```

15 Appendix F

This chapter, together with the Chapters 16 and 17, is the Appendix F required in the LRM, in which all implementation-dependent characteristics of an Ada implementation are described.

15.1 Implementation-Dependent Pragmas

The form, allowed places, and effect of every implementation-dependent pragma is stated in this section.

15.1.1 Predefined Language Pragmas

The form and allowed places of the following pragmas are defined by the language: their effect is (at least partly) implementation-dependent and stated here.

CONTROLLED
has no effect.

ELABORATE
is fully implemented. The Compiler assumes a PRAGMA elaborate, i.e. stores a unit in the library as if a PRAGMA elaborate for a unit u was given, if the compiled unit contains an instantiantion of u (or for a generic program unit in u) and if it is clear that u *must* have been elaborated before the compiled unit. In this case an appropriate information message is given. By this means it is avoided that an elaboration order is chosen which would lead to a PROGRAM_ERROR when elaborating the instantiation.

INLINE
Inline expansion of subprograms is supported with the following restrictions: the subprogram must not contain declarations of other subprograms, tasks, generic units or body stubs. If the subprogram is called recursively only the outer call of this subprogram will be expanded.

INTERFACE
is supported for ASSEMBLER and C. `PRAGMA interface(assembler,...)` provides an interface with the internal calling conventions of the Ada System.

`PRAGMA interface(C,...)` is provided to support the MIPS procedure calling standard.

`PRAGMA interface` should always be used in connection with the `PRAGMA external_name` (see §15.1.2), otherwise the Compiler will generate an internal name that leads to an unsolved reference during linking. These generated names are prefixed with an underline; therefore the user should not use names beginning with an underline.

LIST
is fully implemented.

MEMORY_SIZE
has no effect.

OPTIMIZE
has no effect.

PACK
see §16.1.

PAGE
is fully implemented. Note that form feed characters in the source do not cause a new page in the listing. They are - as well the other format effectors (horizontal tabulation, vertical tabulation, carriage return, and line feed) - replaced by a ˜ character in the listing.

PRIORITY
There are two implementation-defined aspects of this pragma: First, the range of the subtype `priority`, and second, the effect on scheduling (Chapter 14) of not giving this pragma for a task or main program. The range of subtype `priority` is 0 .. 15, as declared in the predefined library package `system` (see §15.3); and the effect on scheduling of leaving the priority of a task or main program undefined by

not giving PRAGMA `priority` for it is the same as if the PRAGMA `priority` 0 had been given (i.e. the task has the lowest priority).

SHARED
is fully supported.

STORAGE_UNIT
has no effect.

SUPPRESS
has no effect, but see §15.1.2 for the implementation-defined PRAGMA `suppress_all`.

SYSTEM_NAME
has no effect.

15.1.2 Implementation-Defined Pragmas

BYTE_PACK
see §16.1.

EXTERNAL_NAME (<string>, <ada_name>)
<ada_name> specifies the name of a subprogram or of an object declared in a library package, <string> must be a string literal. It defines the external name of the specified item. The Compiler uses a symbol with this name in the `call` instruction for the subprogram. The subprogram declaration of <ada_name> must precede this pragma. If several subprograms with the same name satisfy this requirement the pragma refers to that subprogram which is declared last.
Upper and lower cases are distinguished within <string>, i.e. <string> must be given exactly as it is to be used by external routines. This pragma will be used in connection with the pragmas `interface (C)` or `interface (assembler)` (see §15.1.1).

RESIDENT (<ada_name>)

this pragma causes the value of the object to be held in memory and prevents assignments of a value to the object <ada_name> from being eliminated by the optimizer. The following code sequence demonstrates the intended usage of the pragma:

```
...
x : integer;
a : SYSTEM.address;
  ...
BEGIN
  x := 5;
  a := x'ADDRESS;
  do_something (a);  -- let do_something be a non-local
                     -- procedure
                     -- a.ALL will be read in the body
                     -- of do_something
  x := 6;
  ...
```

If this code sequence is compiled with the optimizer on, the statement `x := 5;` will be eliminated because from the point of view of the optimizer the value of x is not used before the next assignment to x. Therefore

```
PRAGMA resident (x);
```

should be inserted after the declaration of x.

This pragma can be applied to all those kinds of objects for which the address clause is supported (cf. §16.5).
It will often be used in connection with the PRAGMA `interface (C, ... )` (see §15.1.4).

SUPPRESS_ALL

causes all the runtime checks described in the LRM(§11.7) to be suppressed; this pragma is only allowed at the start of a compilation before the first compilation unit; it applies to the whole compilation.

15.2 Implementation-Dependent Attributes

The name, type and implementation-dependent aspects of every implementation-dependent attribute is stated in this section.

15.2.1 Language-Defined Attributes

The name and type of all the language-defined attributes are as given in the LRM. We note here only the implementation-dependent aspects.

ADDRESS

If this attribute is applied to an object for which storage is allocated, it yields the address of the first storage unit that is occupied by the object.
If it is applied to a subprogram or to a task, it yields the address of the entry point of the subprogram or task body.
If it is applied to a task entry for which an address clause is given, it yields the address given in the address clause.
For any other entity this attribute is not supported and will return the value **`system.address_zero`**.

IMAGE

The image of a character other than a graphic character (cf. LRM(§3.5.5(11))) is the string obtained by replacing each italic character in the indication of the character literal (given in the LRM(Annex C(13))) by the corresponding upper-case character. For example, `character'image(`*nul*`) = "NUL"`.

MACHINE_OVERFLOWS

Yields always **`false`**.

MACHINE_ROUNDS

Yields always **`false`**.

STORAGE_SIZE

The value delivered by this attribute applied to an access type is as follows:

If a length specification (STORAGE_SIZE, see §16.2) has been given for that type (static collection), the attribute delivers that specified value.
In case of a dynamic collection, i.e. no length specification by STORAGE_SIZE given for the access type, the attribute delivers the number of storage units currently allocated for the collection. Note that dynamic collections are extended if needed.
If the collection manager is used for a dynamic collection the attribute delivers the number of storage units currently allocated for the collection. Note that in this case the number of storage units currently allocated may be decreased by release operations.

The value delivered by this attribute applied to a task type or task object is as follows:
If a length specification (STORAGE_SIZE, see §16.2) has been given for the task type, the attribute delivers that specified value; otherwise, the default value is returned.

15.2.2 Implementation-Defined Attributes

There are no implementation-defined attributes.

15.3 Specification of the Package SYSTEM

The package **system** as required in the LRM(§13.7) is reprinted here with all implementation-dependent characteristics and extensions filled in.

```
PACKAGE system IS

   TYPE designated_by_address IS LIMITED PRIVATE;

   TYPE address IS ACCESS designated_by_address;
   FOR address'storage_size USE 0;

   address_zero : CONSTANT address := NULL;

   TYPE name IS (mips_rtu);
   system_name  : CONSTANT name := mips_rtu;

   storage_unit : CONSTANT := 8;
   memory_size  : CONSTANT := 2 ** 31;
   min_int      : CONSTANT := - 2 ** 31;
   max_int      : CONSTANT := 2 ** 31 - 1;
```

```
max_digits   : CONSTANT := 15;
max_mantissa : CONSTANT := 31;
fine_delta   : CONSTANT := 2.0 ** (-31);
tick         : CONSTANT := 1.0/60.0;

SUBTYPE priority IS integer RANGE 0 .. 15;

FUNCTION "+" (left : address; right : integer) RETURN address;
FUNCTION "+" (left : integer; right : address) RETURN address;
FUNCTION "-" (left : address; right : integer) RETURN address;
FUNCTION "-" (left : address; right : address) RETURN integer;

SUBTYPE external_address IS STRING;
-- External addresses use hexadecimal notation with characters
-- '0'..'9', 'a'..'f' and 'A'..'F'. For instance:
--    "7FFFFFFF"
--    "80000000"
--    "8" represents the same address as "00000008"

FUNCTION convert_address (addr : external_address) RETURN address;
   -- CONSTRAINT_ERROR is raised if the external address ADDR
   -- is the empty string, contains characters other than
   -- '0'..'9', 'a'..'f', 'A'..'F' or if the resulting address
   -- value cannot be represented with 32 bits.

FUNCTION convert_address (addr : address) RETURN external_address;
   -- The resulting external address consists of exactly 8
   -- characters '0'..'9', 'A'..'F'.

TYPE interrupt_number IS RANGE 1 .. 32;

interrupt_vector : ARRAY (interrupt_number) OF address;
-- The mapping of signal numbers to interrupt addresses is
-- defined by this array.

sighup    : CONSTANT := 1;
sigint    : CONSTANT := 2;
sigquit   : CONSTANT := 3;
sigill    : CONSTANT := 4;
sigtrap   : CONSTANT := 5;
sigiot    : CONSTANT := 6;
sigabrt   : CONSTANT := sigiot;
sigemt    : CONSTANT := 7;
sigfpe    : CONSTANT := 8;
sigkill   : CONSTANT := 9;
sigbus    : CONSTANT := 10;
sigsegv   : CONSTANT := 11;
```

```
sigsys    : CONSTANT := 12;
sigpipe   : CONSTANT := 13;
sigalrm   : CONSTANT := 14;
sigterm   : CONSTANT := 15;
sigusr1   : CONSTANT := 16;
sigusr2   : CONSTANT := 17;
sigchld   : CONSTANT := 18;
sigcld    : CONSTANT := sigchld;
sigpwr    : CONSTANT := 19;
sigstop   : CONSTANT := 20;
sigtstp   : CONSTANT := 21;
sigcont   : CONSTANT := 22;
sigttin   : CONSTANT := 23;
sigttou   : CONSTANT := 24;
sigtint   : CONSTANT := 25;
sigxcpu   : CONSTANT := 26;
sigxfsz   : CONSTANT := 27;
sigwinch  : CONSTANT := 28;
sigurg    : CONSTANT := 29;
sigvtalrm : CONSTANT := 30;
sigprof   : CONSTANT := 31;
sigio     : CONSTANT := 32;
sigpoll   : CONSTANT := sigio;

non_ada_error : EXCEPTION;

-- non_ada_error is raised, if some event occurs which does not
-- correspond to any situation covered by Ada, e.g.:
--    illegal instruction encountered
--    error during address translation
--    illegal address

TYPE exception_id IS NEW address;

no_exception_id       : CONSTANT exception_id := address_zero;

-- Coding of the predefined exceptions:
constraint_error_id : CONSTANT exception_id := ... ;
numeric_error_id    : CONSTANT exception_id := ... ;
program_error_id    : CONSTANT exception_id := ... ;
storage_error_id    : CONSTANT exception_id := ... ;
tasking_error_id    : CONSTANT exception_id := ... ;

non_ada_error_id    : CONSTANT exception_id := ... ;

status_error_id     : CONSTANT exception_id := ... ;
mode_error_id       : CONSTANT exception_id := ... ;
```

```
   name_error_id        : CONSTANT exception_id := ... ;
   use_error_id         : CONSTANT exception_id := ... ;
   device_error_id      : CONSTANT exception_id := ... ;
   end_error_id         : CONSTANT exception_id := ... ;
   data_error_id        : CONSTANT exception_id := ... ;
   layout_error_id      : CONSTANT exception_id := ... ;
   time_error_id        : CONSTANT exception_id := ... ;

   no_error_code      : CONSTANT := 0;

   TYPE exception_information
        IS RECORD
           excp_id          : exception_id;
              -- Identification of the exception. The codings of
              -- the predefined exceptions are given above.
           code_addr        : address;
              -- Code address where the exception occured. Depending
              -- on the kind of the exception it may be be address of
              -- the instruction which caused the exception, or it
              -- may be the address of the instruction which would
              -- have been executed if the exception had not occured.
           error_code       : integer;
      END RECORD;

   PROCEDURE get_exception_information
               (excp_info : OUT exception_information);
      -- The subprogram get_exception_information must only be called
      -- from within an exception handler BEFORE ANY OTHER EXCEPTION
      -- IS RAISED. It then returns the information record about the
      -- actually handled exception.
      -- Otherwise, its result is undefined.

   TYPE exit_code IS NEW integer;
   error          : CONSTANT exit_code := 1;
   success        : CONSTANT exit_code := 0;

   PROCEDURE set_exit_code (val : exit_code);
      -- Specifies the exit code which is returned to the
      -- operating system if the Ada program terminates normally.
      -- The default exit code is 'success'. If the program is
      -- abandoned because of an exception, the exit code is
      -- 'error'.

PRIVATE
   -- private declarations
END system;
```

15.4 Restrictions on Representation Clauses

See Chapter 16 of this manual.

15.5 Conventions for Implementation-Generated Names

There are implementation generated components but these have no names. (cf. §16.4 of this manual).

15.6 Expressions in Address Clauses

See §16.5 of this manual.

15.7 Restrictions on Unchecked Conversions

The implementation supports unchecked type conversions for all kinds of source and target types with the restriction that the target type must not be an unconstrained array type. The result value of the unchecked conversion is unpredictable, if

```
target_type'SIZE > source_type'SIZE
```

15.8 Characteristics of the Input-Output Packages

The implementation-dependent characteristics of the input-output packages as defined in the LRM (Chapter 14) are reported in Chapter 17 of this manual.

15.9 Requirements for a Main Program

A main program must be a parameterless library procedure. This procedure may be a generic instantiation; the generic procedure need not be a library unit.

15.10 Unchecked Storage Deallocation

The generic procedure unchecked_deallocation is provided; the effect of calling an instance of this procedure is as described in the LRM(§13.10.1).

The implementation also provides an implementation-defined package collection_manager, which has advantages over unchecked deallocation in some applications.

Unchecked deallocation and operations of the collection_manager can be combined as follows:

- collection_manager.reset can be applied to a collection on which unchecked deallocation has also been used. The effect is that storage of all objects of the collection is reclaimed.
- After the first unchecked_deallocation (release) on a collection, all following calls of release (unchecked deallocation) until the next reset have no effect, i.e. storage is not reclaimed.
- after a reset a collection can be managed by mark and release (resp. unchecked_deallocation) with the normal effect even if it was managed by unchecked_deallocation (resp. mark and release) before the reset.

15.11 Machine Code Insertions

A package machine_code is not provided and machine code insertions are not supported.

15.12 Numeric Error

The predefined exception numeric_error is never raised implicitly by any predefined operation; instead the predefined exception constraint_error is raised.

16 Appendix F: Representation Clauses

In this chapter we follow the section numbering of Chapter 13 of the LRM and provide notes for the use of the features described in each section.

16.1 Pragmas

PACK

As stipulated in the LRM(§13.1), this pragma may be given for a record or array type. It causes the Compiler to select a representation for this type such that gaps between the storage areas allocated to consecutive components are minimized. For components whose type is an array or record type the PRAGMA PACK has no effect on the mapping of the component type. For all other component types the Compiler will choose a representation for the component type that needs minimal storage space (packing down to the bit level). Thus the components of a packed data structure will in general not start at storage unit boundaries.

BYTE_PACK

This is an implementation-defined pragma which takes the same argument as the predefined language PRAGMA PACK and is allowed at the same positions. For components whose type is an array or record type the PRAGMA BYTE_PACK has no effect on the mapping of the component type. For all other component types the Compiler will try to choose a more compact representation for the component type. But in contrast to PRAGMA PACK all components of a packed data structure will start at storage unit boundaries and the size of the components will be a multiple of **system.storage_unit**. Thus, the PRAGMA BYTE_PACK does not effect packing down to the bit level (for this see PRAGMA PACK).

16.2 Length Clauses

`SIZE`

for all integer, fixed point and enumeration types the value must be <= 32;
for `float` types the value must be = 32 (this is the amount of storage which is associated with these types anyway);
for `long_float` types the value must be = 64 (this is the amount of storage which is associated with these types anyway).
for access types the value must be = 32 (this is the amount of storage which is associated with these types anyway).
If any of the above restrictions are violated, the Compiler responds with a RESTRICTION error message in the Compiler listing.

`STORAGE_SIZE`

Collection size: If no length clause is given, the storage space needed to contain objects designated by values of the access type and by values of other types derived from it is extended dynamically at runtime as needed. If, on the other hand, a length clause is given, the number of storage units stipulated in the length clause is reserved, and no dynamic extension at runtime occurs.

Storage for tasks: The memory space reserved for a task is 10K bytes if no length clause is given. If the task is to be allotted either more or less space, a length clause must be given for its task type, and then all tasks of this type will be allotted the amount of space stipulated in the length clause (the activation of a small task requires about 1.4K bytes). Whether a length clause is given or not, the space allotted is not extended dynamically at runtime.

`SMALL`

there is no implementation-dependent restriction. Any specification for `SMALL` that is allowed by the LRM can be given. In particular those values for `SMALL` are also supported which are not a power of two.

16.3 Enumeration Representation Clauses

The integer codes specified for the enumeration type have to lie inside the range of the largest integer type which is supported; this is the type `integer` defined in package `standard`.

16.4 Record Representation Clauses

Record representation clauses are supported. The value of the expression given in an alignment clause must be 0, 1, 2 or 4. If this restriction is violated, the Compiler responds with a RESTRICTION error message in the Compiler listing. If the value is 0 the objects of the corresponding record type will not be aligned, if it is 1, 2 or 4 the starting address of an object will be a multiple of the specified alignment.

The number of bits specified by the range of a component clause must not be greater than the amount of storage occupied by this component. (Gaps between components can be forced by leaving some bits unused but not by specifying a bigger range than needed.) Violation of this restriction will produce a RESTRICTION error message.

There are implementation-dependent components of record types generated in the following cases :

- If the record type includes variant parts and if it has either more than one discriminant or else the only discriminant may hold more than 256 different values, the generated component holds the size of the record object.
- If the record type includes array or record components whose sizes depend on discriminants, the generated components hold the offsets of these record components (relative to the corresponding generated component) in the record object.

But there are no implementation-generated names (cf. LRM(§13.4(8))) denoting these components. So the mapping of these components cannot be influenced by a representation clause.

16.5 Address Clauses

Address clauses are supported for objects declared by an object declaration and for single task entries. If an address clause is given for a subprogram, package or a task unit, the Compiler responds with a RESTRICTION error message in the Compiler listing.

If an address clause is given for an object, the storage occupied by the object starts at the given address. Address clauses for single entries are supported.

16.6 Change of Representation

The implementation places no additional restrictions on changes of representation.

17 Appendix F: Input-Output

The NAME parameter is a system-dependent parameter that is used for control of external files. It must be a legal RTU pathname conforming to the following syntax:

```
pathname ::= [/] [dirname {/ dirname} /] filename
```

dirname and filename are strings of up to 14 characters length. Any character except ASCII.NUL, ' '(blank), and '/'(slash) may be used.

There are two implementation-dependent types for TEXT_IO:

```
type COUNT is range 0 .. integer'last;
subtype FIELD is integer range 0 .. 512;
```

The line terminator is implemented by the character ASCII.LF, the page terminator by ASCII.FF. There is no character for the file terminator. End of file is deduced from the file size.